Combined Code and Data Minimization Algorithms

March 10, 1995

Mini Conference on Ptolemy

Praveen K. Murthy (UC Berkeley),

Shuvra S. Bhattacharyya (Hitachi America Ltd.),

Edward A. Lee (UC Berkeley)

{murthy,shuvra,eal}@eecs.berkeley.edu

Publications on this material are available on WWW:

`http://ptolemy.eecs.berkeley.edu/papers/PganRpmcDppo/`

Problem Statement

Given an acyclic, multirate SDF graph, want a single appearance schedule that minimizes the amount of data needed for buffering.

Buffering Model:

- **Buffer on every arc in the graph.**
- **The size of the buffer is given by the maximum number of tokens queued on the arc in the schedule.**
- **Total buffering cost given by sum of sizes of individual buffer sizes.**
- **Want to find a schedule that minimizes this cost.**

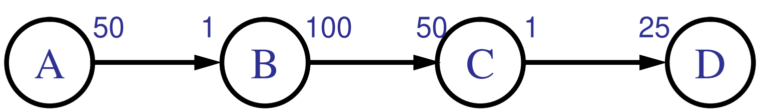
Alternative Buffering Models

Alternative #1: *Naive* single appearance schedules with shared buffers.

- Buffering requirement can be very bad for some graphs.
- Does not handle delays well.
- Latency is maximized.

Alternative #2: Use nested schedules with buffer sharing.

- More awkward to implement.
- Cost function is more complicated.



A (50 B) (100 C) (4 D): Cost = 5000

A (2 (25 (B (2 C))) (2 D)): Cost = 200

Well Ordered Graphs

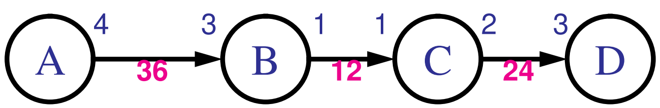
- **A *well-ordered* graph has only one topological sort (i.e., there is a hamiltonian path in the graph).**
- **Problem of computing minimum buffer schedule boils down to computing an optimum nesting of loops.**
- **Done via *dynamic programming* in $O(n^3)$ time:**

$$b[i,j] = MIN_{i \le k < j}\{b[i,k] + b[k+1,j] + c_{ij}[k]\}$$

where $c_{ij}[k] = \dfrac{r_k O_k}{gcd(r_i, \ldots, r_j)}$.

- Note: r_u or $r(u)$ will mean the repetitions of node u.

Example Well Ordered Graph

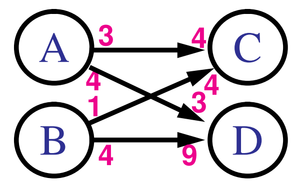


Repetitions vector $r = [9, 12, 12, 8]^T$.

Schedule	Buffering cost
(9A)(12B)(12C)(8D)	72
(3(3A)(4BC))(8D)	37
(3(3A)(4B))(4(3C)(2D))	30

General Acyclic Graphs

- Any *topological sort* of an acyclic graph leads to a set of valid single appearance schedules.
- An acyclic graph can have an exponential number of topological sorts in general: a complete $2n$ node bipartite graph has $(n!)^2$ topological sorts.
- The problem is to pick the topological sort that leads to the best nested schedule when nested optimally using dynamic programming algorithm.



(3 (4 **A**) (3 (4 **B**) **C**)) (16 **D**): 208

(4 (3 **A**) (9 **B**) (4 **D**)) (9 **C**): 120

Two Heuristic Techniques

- We give two heuristic techniques for finding buffer-optimal schedules for acyclic graphs:
  - First technique is a *top-down* approach using min-cuts called *Recursive Partitioning by Minimum Cuts* (RPMC).
    - — Effective for irregular topologies
  - Second technique is a *bottom-up* approach using clustering called *Acyclic Pairwise Grouping of Adjacent Nodes* (APGAN).
    - — Effective for regular topologies
    - — Optimal for a class of graphs

Recursive Partitioning by Min Cuts

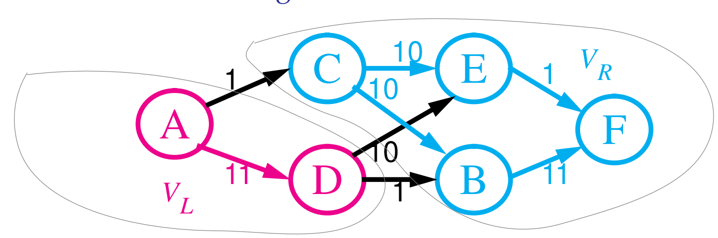
Idea: Find a *cut* of the graph such that

a) All arcs cross the cut in the forward direction.

b) The cut results in fairly even-sized sets.

c) Amount of data crossing the cut is minimized.

Recursively schedule the nodes on the left side of the cut before nodes on the right side of the cut.



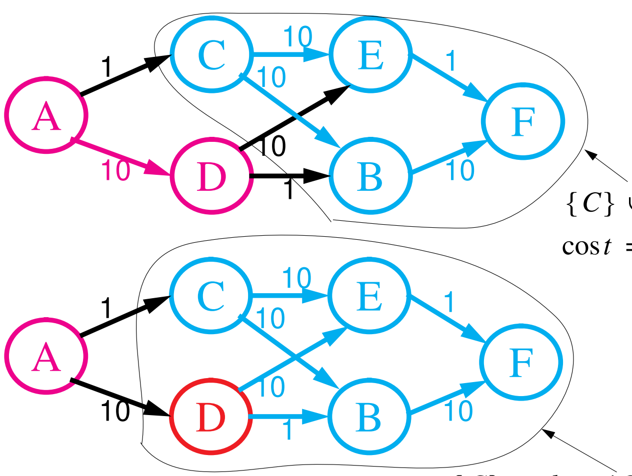
RPMC (cont'd.)

- Splitting the graph where the minimum amount of data is transferred is a *greedy* approach and is not optimal in general.
- Finding the minimum cut such that all of the conditions a,b, and c are satisfied is itself a difficult problem:
  - Methods based on max-flow-min-cut theorem do not work.
  - Graph partitioning when the size of the partition has to be bounded is NP-complete.
- Therefore, a heuristic solution is needed.

A Heuristic for Legal Min Cuts

- Let $V_R(u)$ be the set of nodes consisting of u and its descendents. Let $V_L = V \setminus V_R(u)$.
- This forms a cut satisfying condition (a).
- Perform a local optimization by moving those nodes from V_L that reduce the cost into $V_R(u)$.
- Do this for all nodes u in the graph.
- Repeat above steps to generate cuts obtained by letting $V_L(u)$ be the set of nodes consisting of u and it ancestors, and letting $V_R = V \setminus V_L(u)$.
- Keep the cut with the lowest cost.
- Runs in time $O\left(|V||E| + |V|^2 \bullet \log(|V|)\right)$.

EXAMPLE



$\{C\} \cup desc(C)$, $cost = 12$

$\{C\} \cup desc(C) \cup \{D\}$, $cost = 11$

RPMC Algorithm

- Find heuristic minimum cut of the graph into sets V_L and V_R.
  — The top level schedule is given by

$$S(V) = \left(q_L S\left(V_L\right)\right)\left(q_R S\left(V_R\right)\right) \text{ where } q_i = gcd\{r(v) : v \in V_i\}, i = L, R.$$

- Continue recursively until all nodes have been scheduled.
- Post-process resulting schedule by recomputing an optimum nesting of the loops using dynamic programming algorithm with the lexical ordering generated by RPMC.
- Runs in time $O\left(|V|^3\right)$ for sparse graphs.

Acyclic Pairwise Grouping of Nodes

Idea: Develop a loop hierarchy by clustering two adjacent nodes at each step.

Definition: *Clustering* means combining two or more nodes into one hierarchical node.

- The graph with the hierarchical node instead of the nodes that were clustered is called the *clustered graph*.

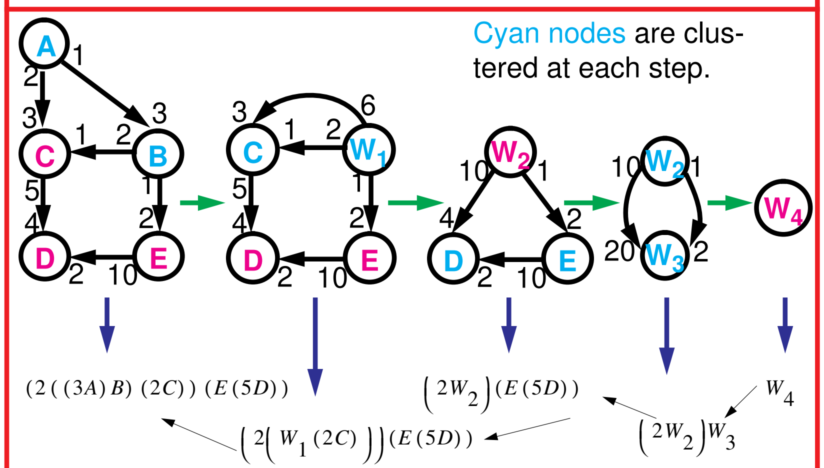
Definition: A *clusterizable* pair of nodes is a pair of nodes that, when clustered, does not cause *deadlock*.

- A sufficient condition for clusterizability: Two nodes are clusterizable if clustering them does not introduce a cycle in the clustered graph.

APGAN Algorithm

- **Cluster two nodes that maximize $gcd\{r(A), r(B)\}$ over all clusterizable pairs $\{A, B\}$.**
- **Continue until only one node is left in the clustered graph**
  - **— This is similar to the Huffman coding algorithm.**
- **After constructing cluster hierarchy, retrace steps to determine the nested schedule.**
- **Post-process the schedule using dynamic programming to generate an optimal nesting for the lexical ordering generated by APGAN.**
- **Runs in time $O\left(|V|^3\right)$ for sparse graphs.**

APGAN in Action



Optimality of APGAN

Definition: The *buffer memory lower bound* for an arc (u, v) is given by

$$BMLB(u, v) = \frac{r(u)\, prod(u, v)}{gcd\{r(u), r(v)\}}$$

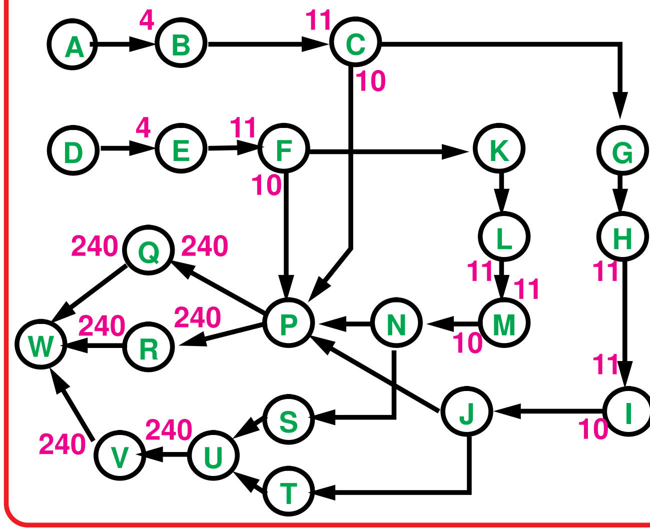
— This represents the least amount of buffering needed on this arc in any single appearance schedule.

Definition: A *BMLB schedule* for an acyclic SDF graph is a single appearance schedule whose buffering cost is equal to the sum of the BMLB costs for each arc.

Theorem: The APGAN algorithm will find a BMLB schedule whenever one exists.

Mobile Satellite Receiver Example

This example is from [Ritz95]:



BMLB = 1540

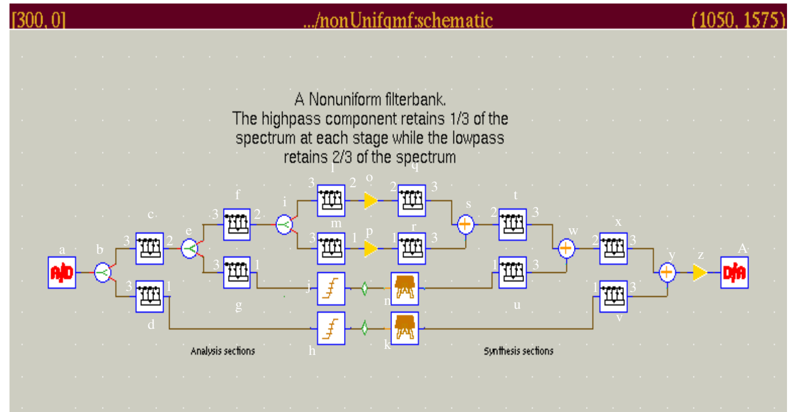
APGAN = 1540

RPMC = 2480

Ritz* = 2040

* Ritz generates a naive single appearance schedule and uses the shared buffer cost.

Non-uniform Filterbank Example



BMLB = 85

RPMC = 128

APGAN = 137

Performance on Practical Examples

Performance of the two heuristics on various acyclic graphs.

System	BMUB	BMLB	APGAN	RPMC	Average Random	Graph size(nodes/ arcs)
Fractional decimation	61	47	47	52	52	26/30
Laplacian pyramid	115	95	99	99	102	12/13
Nonuniform filterbank (1/3,2/3 splits, 4 channels)	466	85	137	128	172	27/29
Nonuniform filterbank (1/3,2/3 splits, 6 channels)	4853	224	756	589	1025	43/47
QMF nonuniform-tree filterbank	284	154	160	171	177	42/45
QMF filterbank (one-sided tree)	162	102	108	110	112	20/22
QMF analysis only	248	35	35	35	43	26/25
QMF Tree filterbank (4 channels)	84	46	46	55	53	32/34
QMF Tree filterbank (8 channels)	152	78	78	87	93	44/50
QMF Tree filterbank (16 channels)	400	166	166	200	227	92/106

Performance on Random Graphs

Performance of the two heuristics on random graphs

RPMC < APGAN	63%
APGAN < RPMC	37%
RPMC < min(2 random)	83%
APGAN < min(2 random)	68%
RPMC < min(4 random)	75%
APGAN < min(4 random)	61%
min(RPMC,APGAN) < min(4 random)	87%
RPMC < APGAN by more than 10%	45%
RPMC < APGAN by more than 20%	35%
APGAN < RPMC by more than 10%	23%
APGAN < RPMC by more than 20%	14%

Conclusion

- Have presented 3 algorithms for joint code and data minimization when synthesizing code from SDF graphs.
- The problem of jointly minimizing code and data boils down to picking an optimal lexical ordering of the nodes and generating an optimal looping hierarchy for that ordering.
- Dynamic programming algorithm generates an optimum looping hierarchy for any given lexical ordering.
- Two heuristics are used to generate lexical orderings:
  - — RPMC: Does well on some practical examples with irregular topologies and on random graphs
  - — APGAN: Does well on a lot of practical examples but not as well on random graphs. It is optimal for a class of graphs.